5

Formatiranje

Kad čitaoci „zavire pod haubu“ našeg koda, želimo da ih impresioniramo urednošću, doslednošću i osećajem za detalje. Želimo da ih urednost ne ostavi ravnodušnim, da izviju obrve dok prolaze kroz module i da shvate da su profesionalci uradili svoj posao. Ako umesto toga vide izmešanu masu koda koja izgleda kao da ju je napisala gomila pijanih mornara, onda će verovatno zaključiti da je ista pažnja posvećena detaljima i svakom drugom aspektu projekta.

Treba da vodite računa da vaš kod bude lepo formatiran. Trebalo bi da odaberete skup jednostavnih pravila koja upravljaju formatiranjem vašeg koda, a zatim da ih dosledno primjenjujete. Ako radite u timu, trebalo bi prihvatiti jedan skup pravila formatiranja kojih svi članovi tima moraju da se pridržavaju. U tome vam može pomoći automatizovana alatka koja primenjuje ta pravila.

Svrha formatiranja

Da budemo jasni, formatiranje koda je *važno*. Previše je važno da bi se ignorisalo i previše je važno pa mu se moramo posvetiti. Formatiranje koda odnosi se na komunikaciju, a komunikacija je od prvorazrednog značaja za profesionalnog programera.

Možda ste mislili da je za profesionalnog progrmera prvo poslovno pravilo „da proradi“. Nadam da vam je ova knjiga do sada razjasnila tu zabludu. Funkcionalnost koju danas stvarate ima dobre šanse da se promeni u sledećem verzijama, ali čitljivost vašeg koda imaće dubok uticaj na sve promene koje će se ikada desiti. Stil kodiranja i čitljivost nastavljaju da utiču na održivost i proširivost koda dugo nakon što je originalna verzija promenjena do neprepoznatljivosti. Vaš stil i disciplina preživljavaju, čak i ako to nije slučaj sa vašim kodom.

Koje su bitne stvari u vezi formatiranja koje nam pomažu da najbolje komuniciramo?

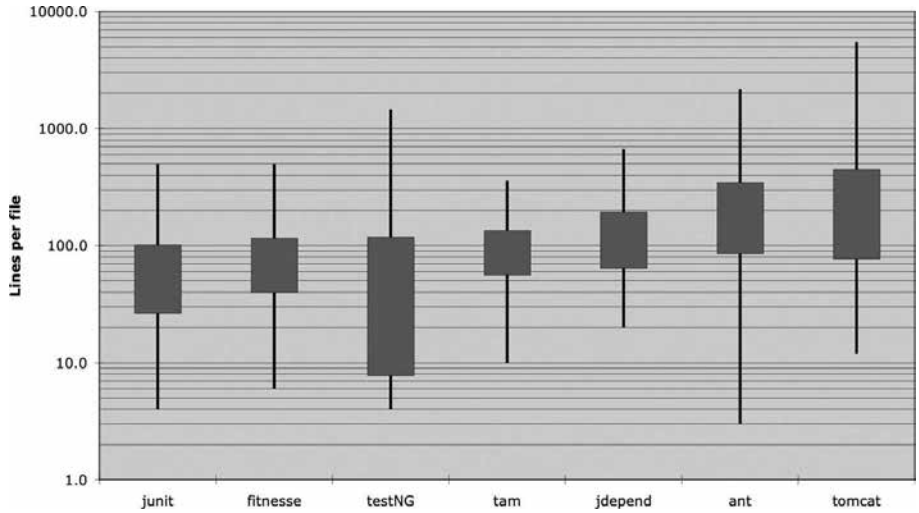
Vertikalno formatiranje

Krenimo od veličine po vertikali. Koliko bi trebalo da bude dugačak izvorni kod? U Javi je veličina koda usko povezana sa veličinom klase. Razgovaraćemo o veličini klasa kada budemo govorili o klasama. Za sada, razmotrimo samo veličinu.

Koliko je velika većina izvornih kodova pisanih u Javi? Otkriva se da postoji ogroman raspon veličina i neke izuzetne razlike u stilu. Slika 5-1 prikazuje neke od tih razlika.

Prikazano je sedam različitih projekata. Junit, FitNesse, testNG, Time and Money, JDepend, Ant i Tomcat. Linije kroz pravougaono polje prikazuju minimalnu i maksimalnu dužinu koda svakog projekta. Polje obuhvata otprilike jednu trećinu (jedna standardna devijacija)[1] dužine koda. Sredina polja je srednja vrednost. Dakle, prosečna veličina datoteke u projektu FitNesse iznosi oko 65 redova koda, a približno jedna trećina datoteka sadrži između 40 i više od 100 redova koda. Najveće polje u FitNesseu je oko 400 redova, a najmanje 6 redova. Imajte na umu da se radi o prikazu sa logaritamskom skalom, tako da mala razlika u poziciji po vertikali znači vrlo veliku razliku u apsolutnoj veličini.

1. Okvir prikazuje sigmu/2 iznad i ispod srednje vrednosti. Znam da raspodela dužina datoteke nije normalna, pa i standardna devijacija nije matematički precizna, ali ovde ne pokušavamo da budemo precizni. Pokušavamo samo da shvatimo o čemu se radi.



Slika 5-1 Raspodela dužina datoteke na LOG skali (visina pravougaonika = sigma)

Junit, FitNesse i Time and Money sastoje se od relativno malih datoteka. Nijedna nije veća od 500 redova, a većina njih je kraća od 200 redova. Tomcat i Ant, s druge strane, imaju neke datoteke koje su duge po nekoliko hiljada redova, a blizu polovine su sa više od 200 redova koda.

Šta nam to znači? Čini se da je moguće izgraditi značajne sisteme (FitNesse je blizu 50.000 redova) od datoteka koje su obično duge 200 redova, sa gornjom granicom koja iznosi 500 redova. Iako ovo ne bi trebalo da bude važno pravilo, trebalo bi ga smatrati vrlo poželjnim. Male datoteke je obično lakše razumeti nego velike.

Novinska metafora

Razmislite o dobro napisanom novinskom članku. Pročitali ste ga vertikalno. Na vrhu vas očekuje naslov koji će vam reći o čemu se radi i omogućava vam da odlučite da li je to nešto što želite da pročitate. Prvi pasus daje sažetak celog članka, skrivajući sve detalje dok vam daje široku sliku. Kako nastavljate da čitate dalje, detalji se pojavljuju sve dok ne dobijete uvid u sve datume, imena, citate, tvrdnje i druge sitnice.

Želeli bismo da izvorni kod bude poput novinskog članka. Naziv treba da bude jednostavan, ali objašnjavajući. Ime bi samo po sebi trebalo da bude dovoljno da nam kaže da li smo u pravom modulu ili ne. Delovi na vrhu izvornog koda treba da pružaju koncepte i algoritme visokog nivoa. Pojedinosti bi trebalo da se povećavaju kako nastavljamo sa čitanjem, dok na kraju ne pronađemo najniže nivoe funkcija i detalje izvornog koda.

Novine su sastavljene od puno članaka; većina njih su vrlo kratki. Neki su malo duži, ali vrlo malo njih je sa toliko teksta da bi mogli da zauzmu celu stranicu. Zbog toga su novine *upotrebljive*. Da su novine samo jedna duga priča koja sadrži neorganizovanu kolekciju činjenica, datuma i imena, mi ih jednostavno ne bismo čitali.

Vertikalna otvorenost između pojmova

Skoro ceo kod se čita s leva na desno i odozgo na dole. Svaki red koda predstavlja izraz ili uslov, a svaka grupa redova predstavlja potpunu misao. Te misli treba međusobno odvojiti praznim redovim.

Razmotrimo, na primer, listing 5-1. Postoje prazni redovi koji razdvajaju deklaraciju paketa, uvoz(e) i svaku od funkcija. Ovo krajnje jednostavno pravilo ima dubok uticaj na vizuelni izgled koda. Svaki prazan red je vizuelni znak koji identifikuje novi i zasebni koncept. Dok prolazite kroz listing, pogled vam se lepi na prvi red koji sledi iza praznog reda.

Listing 5-1
BoldWidget.java

```
package fitnesse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
  public static final String REGEXP = "'''.+?'''";
  private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
    Pattern.MULTILINE + Pattern.DOTALL
  );

  public BoldWidget(ParentWidget parent, String text) throws Exception {
    super(parent);
    Matcher match = pattern.matcher(text);
    match.find();
    addChildWidgets(match.group(1));
  }

  public String render() throws Exception {
    StringBuffer html = new StringBuffer("<b>");
    html.append(childHtml()).append("</b>");
    return html.toString();
  }
}
```

Izbacivanje ovih praznih redova, kao u listingu 5-2, ima izuzetan uticaj na čitljivost koda.

Listing 5-2
BoldWidget.java

```
package fitnesse.wikitext.widgets;
import java.util.regex.*;
public class BoldWidget extends ParentWidget {
  public static final String REGEXP = "'''.+?'''";
  private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
    Pattern.MULTILINE + Pattern.DOTALL);
  public BoldWidget(ParentWidget parent, String text) throws Exception {
    super(parent);
    Matcher match = pattern.matcher(text);
    match.find();
    addChildWidgets(match.group(1));}
  public String render() throws Exception {
    StringBuffer html = new StringBuffer("<b>");
    html.append(childHtml()).append("</b>");
    return html.toString();
  }
}
```

Ovaj utacaj je još izraženiji kada niste fokusirani. U prvom primeru nameću vam se različite grupe redova, dok drugi primer izgleda kao zbrka. Razlika između ova dva listinga je samo zbog vertikalne otvorenosti.

Vertikalna gustina

Ako otvorenost razdvaja koncepte, tada vertikalna gustina podrazumeva blisku povezanost. Dakle, redovi koda koji su usko povezani treba da izgledaju vertikalno gusti. Obratite pažnju kako beskorisni komentari u listingu 5-3 prekidaju blisku povezanost dve instance.

Listing 5-3
public class ReporterConfig {

```
  /**
   * The class name of the reporter listener
   */
  private String m_className;
  /**
   * The properties of the reporter listener
   */
  private List<Property> m_properties = new ArrayList<Property>();
  public void addProperty(Property property) {
    m_properties.add(property);
  }
```